Escola de Matemática Aplicada

Aula Prática 1 - Álgebra Linear Numérica

Aluno: *Gustavo Murilo Cavalcante Carvalho*
Programa: *Bacharelado em Matemática Aplicada*
Disciplina: *Álgebra Linear Numérica*
Professor: *Antonio Carlos Saraiva Branco*
Email: *gustavomurilo012@gmail.com*

Data: *21 de março de 2024*

Sumário

Problema 1

Teste a função dada usando algumas matrizes quadradas A e respectivos vetores b. Use exemplos dos quais você saiba a resposta para verificar se a função realmente está funcionando corretamente.

Solução:

Considere as seguintes matrizes e vetores:

$$M_1 = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix},\ M_2 = \begin{bmatrix} 1/5 & 1/3 & 1 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix},\ v_1 = \begin{bmatrix} x \\ y \end{bmatrix},\ v_2 = \begin{bmatrix} i \\ j \\ k \end{bmatrix},\ s_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix},\ s_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}$$

A partir disso montamos sistemas lineares que tem solução, afinal eles foram construídos para isso. Seguem as soluções dos sistemas bem como às decomposições LU das matrizes.

Da equação 1:

$$M_1 * v_1 = s_1 \tag{1}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \implies \begin{bmatrix} 1 & 3 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \tag{2}$$

$$\implies \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \end{bmatrix} = v_1 \tag{3}$$

Bem como, $M_1 = L_1 U_1$

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 0 & -2 \end{bmatrix} \tag{4}$$

Ao passar estes parâmetros para a função "Gaussian_Elimination_1", verificamos que o resultado é o esperado:

```
>> disp(M1)
1       3
2       4

>> disp(s1)
0
2
```

```
>> [v1, C1] = Gaussian_Elimination_1(M1, s1)
v1 =
     3
    -1

C1 =
     1     3
     2    -2
```

Da equação 2:

$$M_2 * v_2 = s_2 \tag{5}$$

$$\begin{bmatrix} 1/5 & 1/3 & 1 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} \implies \begin{bmatrix} 1/5 & 1/3 & 1 \\ 0 & 2/3 & 5 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix} \tag{6}$$

$$\implies \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \\ 1 \end{bmatrix} = v_2 \tag{7}$$

Decompondo a matriz, $M_2 = L_2U_2$

$$\begin{bmatrix} 1/5 & 1/3 & 1 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/5 & 1/3 & 1 \\ 0 & 2/3 & 5 \\ 0 & 0 & -1 \end{bmatrix} \tag{8}$$

Utilizando a função "Gaussian_Elimination_1" para esse sistema, obtemos novamente o resultado correto, tanto para a solução do sistema, quanto para a decomposição LU.

```
>> disp(M2)
    0.2000   0.3333   1.0000
   -1.0000  -1.0000        0
         0        0  -1.0000

>> disp(s2)
    1
   -2
   -1
```

```
>> [v2, C2] = Gaussian_Elimination_1(M2, s2)
v2 =
    5.0000
   -3.0000
    1.0000
C2 =
    0.2000    0.3333    1.0000
   -5.0000    0.6667    5.0000
         0         0   -1.0000
```

Problema 2

Agora teste com a matriz $A_1 = \begin{bmatrix} 1 & -2 & 5 & 0 \\ 2 & -4 & 1 & 3 \\ -1 & 1 & 0 & 2 \\ 0 & 3 & 3 & 1 \end{bmatrix}$ e com o vetor $b1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Solução:

Primeiro criamos as variáveis A_1 e b_1:

```
>> disp(A1)
 1    -2     5     0
 2    -4     1     3
-1     1     0     2
 0     3     3     1
```

```
>> disp(b1)
1
0
0
0
```

Então usamos a função "Gaussian_Elimination_1" para achar a decomposição LU de A e achar a solução do sistema linear $A_1x = b_1$. Como podemos ver abaixo, a função falha em ambas as tarefas

```
>> [x, C] = Gaussian_Elimination_1(A1, b1)

x =          C =
   NaN             1    -2     5     0
   NaN             2     0    -9     3
   NaN            -1  -Inf  -Inf   Inf
   NaN             0   Inf   NaN   NaN
```

Isso acontece porque o algorítimo dessa função leva a uma divisão por 0 durante o escalonamento, o que é uma impossibilidade e, por isso, deveria ser evitada.

Para entender melhor o que aconteceu, farei o escalonamento da matriz A seguindo o algorítimo da função "Gaussian_Elimination_1", até encontrar o problema ocorrido.

Dado o sistema $A_1x = b_1$, formamos uma matriz completa (também chamada de matriz aumentada) da forma $[A|b]$ e escalonamos essa matriz, alcançando uma matriz $[U|Lb]$. Usando o algorítimo dado, não podemos trocar linhas, a única ação para escalonar a matriz é subtrair das linhas de baixa uma parte das linhas de cima. Fazendo isso:

$$\begin{bmatrix} 1 & -2 & 5 & 0 \\ 2 & -4 & 1 & 3 \\ -1 & 1 & 0 & 2 \\ 0 & 3 & 3 & 1 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{9}$$

Escalonando $[A|b]$, como a função faz:

$$\begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 2 & -4 & 1 & 3 & | & 0 \\ -1 & 1 & 0 & 2 & | & 0 \\ 0 & 3 & 3 & 1 & | & 0 \end{bmatrix} \begin{matrix} L_3 \leftarrow L_2 - \frac{2}{1}L_1 \\ \\ L_4 \leftarrow L_3 - \frac{-1}{1}L_1 \end{matrix} \begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 0 & 0 & -9 & 3 & | & -2 \\ 0 & -1 & 5 & 2 & | & 1 \\ 0 & 3 & 3 & 1 & | & 0 \end{bmatrix} \tag{10}$$

Não havendo instrução acerca de um 0 no lugar do pivô, as instruções levariam às seguintes operações entre linhas:

$$\begin{aligned} L_3 &\leftarrow L_3 - \frac{-1}{0}L_2 \\ L_4 &\leftarrow L_4 - \frac{3}{0}L_2 \end{aligned} \tag{11}$$

Há aí uma impossibilidade, uma divisão por 0. Esse evento acarreta problemas inesperados. No MATLAB, se $k \in \mathbb{R}\setminus\{0\}$, então $\frac{k}{0}$ a é interpretado como infinito e é representado por Inf, já para $k = 0$ temos que $\frac{0}{0}$ é representado por NaN o que significa "Not a Number".

Problema 3

Modifique a função dada trocando linhas quando no início da iteração j o elemento na posição (j, j) é nulo. Chame esta nova função de Gaussian˙Elimination˙2 e teste-a com a matriz A_1 e o vetor b_1 dados.

Agora teste-a com a matriz $A_2 = \begin{bmatrix} 0 & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$ e o vetor $b_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

Solução:

Alterações na função:

Na primeira função, "Gaussian_Elimination_1", existem 2 loops aninhados. Iremos atualizar essa função, criando a função "Gaussian_Elimination_2". Abaixo há uma indicação da a parte do código que é diferente da primeira função. A função completa pode ser encontrada na seção dedicada para ela.

```
for j=1:(n-1)
  % Aqui alteramos a função, para que linhas sejam trocadas no caso de existir um
  % 0 no lugar do pivô, possibilitando que o escalonamento continue.
  for i=(j+1):n
    % Essa parte é igual a de Gaussian_Elimination_1.
    % Escalonamos a matriz aumentada C e nas linhas que não serão mais usadas,
    % armazenamos os elementos das matrizes L e U da decomposição de A.
  end
end
```

Na parte indicada, há o bloco de código abaixo. Ele está devidamente comentado, mas irei dar mais alguns esclarecimentos.

```
function [x, C, P] = Gaussian_Elimination_2(A, b)

...

  % Se, em vez disso, tivermos um 0, trocamos a linha j por uma linha k,
  % sendo esta a primeira linha sem 0 na posição de pivô.
  if C(j, j) == 0
    % Acha a distância entre j e os elementos abaixo dele que têm valor 0.
    dist = find(C(j+1:n,j));

    % Caso onde a matriz A não é singular. Não existe decomposição LU.
    if dist(1) == 0
      error("O sistema é indeterminado");
```

```
24         end
25 
26         % Atribui a k o índice da linha a ser trocada com j.
27         k = dist(1) + j;
28 
29         % Há a troca das linhas k e j da matriz de permutação na decomposição PLU.
30         P([j k], :) = P([k j], :);
31 
32         % Troca os elementos das linhas k e j da matriz completa [A|b]
33         % a partir da coluna j, pois antes disso todos são 0.
34         C([j k], j:n+1) = C([k j], j:n+1);
35     end
```

Sobre o processo de escalonamento de uma matriz $n \times n$, temos no máximo $n-1$ etapas, pois zeramos os números abaixo do pivô em cada linha com exceção da última, na qual isso não é possível. Em todas as funções acerca da eliminação gaussiana, da 1 à 4, vamos considerar que j é a linha que estamos zerando os números abaixo do pivô em cada etapa, logo é a j-ésima etapa do escalonamento. Essa consideração é válida para todo o documento.

A função find, na linha 19, retorna os índices não nulos, ela é aplicada na parte da coluna j que ainda precisa ser escalonada. Seja $C_{n \times n+1} = [A|b]$, sendo $A_{n \times n}$, $b \in \mathbb{R}^n$. Como dito anteriormente, temos que j é o índice de uma linha qualquer de C exceto a última, ou seja, $j \in [\![1, n-1]\!]$. Também sabemos que as linhas acima de j (de 1 a $j-1$) já passaram pelo processo de escalonamento, por isso o comando só deve verificar as linhas restantes, ou seja, os elementos de C nas posições (i, j) com $i \in [\![j+1, n]\!]$.

Outra parte que merece explicação está na linha 34. Nessa linha há a troca das colunas k e j. No MATLAB é possível fazer isso de forma bem simples manipulando os índices da matriz C. Para trocar as linhas j e k, basta passar para os índices correspondentes as linhas k e j, o que seria

```
C([j k],:) = C([k j],:);
```

Contudo, não precisamos trocas as linhas por completo, pois na j-ésima etapa, as linhas acima de j não serão mais alteradas, bem como as colunas antes de j também não serão, afinal elas só tem 0 nas posições abaixo da linha j e é claro que $\alpha \dot{0} + \beta 0 = 0$. Logo, é certo que terão valor 0 os elementos de índices (f, g) com $g \in [\![1, j-1]\!]$ e $f > g$. Como esses valores podem ser ignorados, ao trocar as linhas só precisamos incluir os elementos da coluna j à coluna $n+1$ de C.

Testes com A_1 e b_1:

```
>> [x, C, P] = Gaussian_Elimination_2(A1, b1)

    x =            C =                            P =
      -0.3248          1    -2     5     0           1     0     0     0
      -0.1709          2    -1     5     2           0     0     1     0
       0.1966         -1     0    -9     3           0     1     0     0
      -0.0769          0    -3    -2    13           0     0     0     1
```

Escalonando $[A_1|b_1]$, como a função faz

$$\begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 2 & -4 & 1 & 3 & | & 0 \\ -1 & 1 & 0 & 2 & | & 0 \\ 0 & 3 & 3 & 1 & | & 0 \end{bmatrix} \begin{matrix} L_3 \leftarrow L_2 - \frac{2}{1}L_1 \\ \\ L_4 \leftarrow L_3 - \frac{-1}{1}L_1 \end{matrix} \begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 0 & 0 & -9 & 3 & | & -2 \\ 0 & -1 & 5 & 2 & | & 1 \\ 0 & 3 & 3 & 1 & | & 0 \end{bmatrix} \tag{12}$$

Trocamos as linhas 2 e 3 e continuamos o escalonamento:

$$\begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 0 & -1 & 5 & 2 & | & 1 \\ 0 & 0 & -9 & 3 & | & -2 \\ 0 & 3 & 3 & 1 & | & 0 \end{bmatrix} L_4 \leftarrow L_4 - \frac{3}{-1}L_2 \begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 0 & -1 & 5 & 2 & | & 1 \\ 0 & 0 & -9 & 3 & | & -2 \\ 0 & 0 & 18 & 7 & | & 3 \end{bmatrix} \tag{13}$$

$$\begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 0 & -1 & 5 & 2 & | & 1 \\ 0 & 0 & -9 & 3 & | & -2 \\ 0 & 0 & 18 & 7 & | & 3 \end{bmatrix} L_4 \leftarrow L_4 - \frac{18}{-9}L_3 \begin{bmatrix} 1 & -2 & 5 & 0 & | & 1 \\ 0 & -1 & 5 & 2 & | & 1 \\ 0 & 0 & -9 & 3 & | & -2 \\ 0 & 0 & 0 & 13 & | & -1 \end{bmatrix} \tag{14}$$

Verificamos (na matriz completa após o escalonamento $[U|L^{-1}b]$) que a função retorna a matriz U correta. Olhando para os indices das eliminações obtemos os elementos de L, por exemplo $(L_a - \frac{\alpha}{\beta}L_b)$ indica que o elemento $L(a,b) = \frac{\alpha}{\beta}$. Sabendo disto, vemos que a matriz L retornada pela função é coerente. Por fim multiplicamos a matriz a fim de verificar a igualdade, disso confirmamos que a função retornou uma fatoração LU correta de A_1.

$$L_1U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -3 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 5 & 0 \\ 0 & -1 & 5 & 2 \\ 0 & 0 & -9 & 3 \\ 0 & 0 & 0 & 13 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 5 & 0 \\ 2 & -4 & 1 & 3 \\ -1 & 1 & 0 & 2 \\ 0 & 3 & 3 & 1 \end{bmatrix} = A_1 \tag{15}$$

A solução do sistema também está correta, segue a verificação disso:

$$A_1x = L_1U_1x = b_1 \implies U_1x = L_1^{-1}b_1 = c_1 \tag{16}$$

$$\implies \begin{bmatrix} 1 & -2 & 5 & 0 \\ 0 & -1 & 5 & 2 \\ 0 & 0 & -9 & 3 \\ 0 & 0 & 0 & 13 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ l \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -2 \\ -1 \end{bmatrix} \tag{17}$$

$$\implies \begin{bmatrix} i \\ j \\ k \\ l \end{bmatrix} = \begin{bmatrix} -38/117 \\ -20/117 \\ 23/117 \\ -1/13 \end{bmatrix} \approx \begin{bmatrix} -0.3248 \\ -0.1709 \\ 0.1966 \\ -0.0769 \end{bmatrix} = x \tag{18}$$

Testes com A_2 e b_2:

Primeiro criamos as variáveis para armazenar as matrizes, e então fazemos o teste.

```
>> disp(b2)
     1
     0
     0
```

```
>> disp(A2)
           0       1e-20           1
       1e-20           1           1
           1           2           1
```

```
>> [x, C, P] = Gaussian_Elimination_2(A2,b2)

x =               C =                                   P =

   -1e+20            1e-20         1         1              0     1     0
        0                0     1e-20         1              1     0     0
        1            1e+20    -1e+40     1e+40              0     0     1
```

Fazendo a eliminação gaussiana como a função. Para isso trocamos as linhas na etapa 1 e seguimos o escalonamento.

$$\begin{bmatrix} 0 & 10^{-20} & 1 & | & 1 \\ 10^{-20} & 1 & 1 & | & 0 \\ 1 & 2 & 1 & | & 0 \end{bmatrix} L_1 \leftrightarrow L_2 \begin{bmatrix} 10^{-20} & 1 & 1 & | & 0 \\ 0 & 10^{-20} & 1 & | & 1 \\ 1 & 2 & 1 & | & 0 \end{bmatrix} \tag{19}$$

$$\begin{bmatrix} 10^{-20} & 1 & 1 & | & 0 \\ 0 & 10^{-20} & 1 & | & 1 \\ 1 & 2 & 1 & | & 0 \end{bmatrix} L_3 \leftarrow L_3 - \frac{1}{10^{-20}} L_1 \begin{bmatrix} 10^{-20} & 1 & 1 & | & 0 \\ 0 & 10^{-20} & 1 & | & 1 \\ 0 & 2 - 10^{20} & 1 - 10^{20} & | & 0 \end{bmatrix} \tag{20}$$

Em operações que envolvem uma grande diferença de magnitude, o MATLAB acaba considerando alguns valores como desprezíveis (os considera 0). Podemos ver isso nos seguintes exemplos:

```
>> disp(2 - power(10,20))
   -1e+20
```

```
>> disp(power(10,40) - power(10,20))
   1e+40
```

Levando isso em consideração para o resto do escalonamento:

$$\begin{bmatrix} 10^{-20} & 1 & 1 & | & 0 \\ 0 & 10^{-20} & 1 & | & 1 \\ 0 & -10^{20} & -10^{20} & | & 0 \end{bmatrix} L_3 \leftarrow L_3 - \frac{-10^{20}}{10^{-20}} L_1 \tag{21}$$

$$\begin{bmatrix} 10^{-20} & 1 & 1 & | & 0 \\ 0 & 10^{-20} & 1 & | & 1 \\ 0 & 0 & 10^{40} & | & 10^{40} \end{bmatrix}$$

O resultado obtido certamente é errôneo, afinal os arredondamentos feitos durante o escalonamento foram bem grosseiro, o que distancia o resultado obtido do resultado correto.

Problema 4

Modifique a função do item 3 para escolher o maior pivô em módulo quando no início da iteração j o elemento na posição (j, j) é nulo. Chame esta nova função de Gaussian˙Elimination˙3 e teste-a com a matriz A_2 e o vetor b_2 dados. Agora com a matriz $A_3 = \begin{bmatrix} 10^{-20} & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$ e o vetor $b_3 = b_2$.

Solução:

Alterações na função:

```
7  function [x, C, P] = Gaussian_Elimination_3(A, b)
```

...

```
17   if C(j, j) == 0
18     % Atribui a dist o índice do elemento com maior valor absoluto dentre
19     % os elementos da coluna j abaixo da linha j.
20     [maior_pivo, dist] = max(abs(C(j+1:n,j)));
21 
22     % Caso onde a matriz A não é singular. Não existe decomposição LU.
23     if maior_pivo == 0
24       error("O sistema é indeterminado");
25     end
26 
27     % Atribui a k o índice da linha com maior pivo em módulo.
28     k = dist(1) + j;
```

A primeira alteração a ser comentada é na linha 20. Agora procuramos dentre os candidatos a pivô o maior em módulo, por isso passamos o intervalo de interesse para a função abs(). Isso é passado para a função max() que retorna dois valores, eles são respectivamente o valor do maior elemento e o seu índice.

A outra alteração, essa menos pertinente, está na linha 23. Nesta linha verificamos se haverá uma ou mais linhas de 0 ao fim do escalonamento, ou seja, se A inversível. Ao perceber que esse é o caso, a função retorna o erro e para a execução.

Teste com A_2 e b_2

```
>> [x, C, P] = Gaussian_Elimination_3(A2, b2)

x =           C =                          P =

     1            1      2      1             1     0     0
    -1        1e-20      1      1             0     1     0
     1            0  1e-20      1             0     0     1
```

Os resultados "Gaussian_Elimination_3" são diferentes da função anterior, contudo eles ainda diferem da solução exata. Ao verificar decomposição PLU dada, vemos que ela não é correta, pois $L_1 U_2 = A_2$.

$$L_2 U_2 = \begin{bmatrix} 1 & 0 & 0 \\ 10^{-20} & 1 & 0 \\ 0 & 10^{-20} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 10^{-20} & 2 * 10^{-20} + 1 & 10^{-20} + 1 \\ 0 & 10^{-20} & 10^{-20} + 1 \end{bmatrix} \neq A_2 \quad (22)$$

Contudo, considerando os arredondamentos que faz o MATLAB (que no caso parecem aceitáveis), temos que a solução é válida, dada que no programa é obtido o resultado esperado:

```
>> disp(A2 * x)
    1
    0
    0

>> disp((eye(3) + tril(C, -1))*triu(C))
    1          2        1
    1e-20      1        1
    0      1e-20        1
```

Esta quando multiplicada pela matriz de permutação resulta em A_2.

Teste com A_3 e b_3

```
>> [x, C, P] = Gaussian_Elimination_3(A3, b3)

    x =         C =                                 P =

        0           1e-20    1e-20          1           1      0      0
       -1               1        1          0           0      1      0
        1           1e+20        1     -1e+20           0      0      1
```

Verificando as saídas, vemos que a solução x é inválida, pois o vetor resultante de A_3x não é b_3.

```
>> disp(A3 * x)
 1
 0
-1
```

Quanto à decomposição PLU, onde $P = I$, vemos que a multiplicação das matrizes triangulares, embora resultem em algo similar, não resulta em A_3, pois o elemento de indice $(3, 3)$ que deveria ser 1 é 0.

```
>> disp((eye(3) + tril(C, -1)) * triu(C))
    1e-20       1e-20           1
    1e-20           1           1
        1           2           0
```

Problema 5

Modifique a função do item 4 para escolher sempre o maior pivô em módulo no início da iteração j independente do elemento na posição (j, j) ser nulo ou não. Nessa função, retorne também a matriz de permutação P utilizada. Chame esta nova função de Gaussian˙Elimination˙4 e teste-a com a matriz A_3 e o vetor b_3 dados.

Solução:

Alterações na função:

```
function [x, C, P] = Gaussian_Elimination_4(A, b)
```

...

```
for j = 1:(n-1)
```

... % Bloco com comentários

```
  [maior_pivo, dist] = max(abs(C(j:n,j)));

  % Caso onde a matriz A não é singular. Não existe decomposição LU.
  if maior_pivo == 0
    error("O sistema é indeterminado");
  end

  % Atribui a k o índice da primeira linha abaixo de j com 0 abaixo de j.
  k = dist(1) + j - 1;

  % Há a troca das linhas k e j da matriz de permutação na decomposição PLU.
  P([j k], :) = P([k j], :);

  % Há a troca dos elementos das linhas k e j da matriz completa [A|b]
  % a partir da coluna j, pois antes disso todos são 0.
  C([j k], j:n+1) = C([k j], j:n+1);

  % Processo de decomposição LU de A.
  % ...
```

A principal alteração aqui foi deixar de verificar se há 0 no lugar do pivô (retirar o if) e trocar sempre que for vantajoso. Essa técnica é chamada de pivoteamento parcial, nela para cada etapa do escalonamento deixamos os maiores números em módulo como pivôs, o que é vantajoso para o computador que tem limitações com os números de ponto flutuante.

Outra observação sobre essa versão é sobre a linha 20, nota-se que os índices da matriz C são diferentes. Passamos a incluir o pivô atual na comparação de maior pivô, nos outros casos não era necessário, pois esse número só seria trocado caso fosse 0, logo é menor que qualquer coisa em módulo.

Como consideramos a linha do pivô atual, temos que subtrair 1 da conta ao atribuir o valor de k, para obtermos o índice referente ao maior valor para o pivô.

Teste com A_3 e b_3

```
>> [x, C, P] = Gaussian_Elimination_4(A3, b3)

x =        C =                                P =

    1              1        2        1           0     0     1
   -1          1e-20        1        1           0     1     0
    1          1e-20   -1e-20        1           1     0     0
```

Verificando a solução:

$$\begin{bmatrix} 10^{-20} & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 10^{-20} \\ 0 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = b_3 \tag{23}$$

Fazendo essa multiplicação manualmente, vemos que a solução não é totalmente correta. Ao fazê-la no MATLAB, acontece uma série de truncamentos e aproximações que fazem com que essa solução funcione no MATLAB.

```
>> disp (A3 * x)
     1
     0
     0
```

A seguir está um exemplo dessas aproximações:

```
>> power(10,-20) + (1 - 1) == 0
ans =
  logical
   0

>> (power(10,-20) + 1) - 1 == 0
ans =
  logical
```

```
    1
```

Acerda da decomposição PLU, essa função, utilizando o método do pivoteamento parcial, também é superio que a anterior. Verificamos no programa que $A_3 = PL_3U_3*$ é verdade para as matrizes retornadas pela função.

```
>> disp(P * (eye(3) + tril(C, -1)) * triu(C))
    1e-20   1e-20       1
    1e-20       1       1
        1       2       1
```

Verificando manualmente:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 10^{-20} & 1 & 0 \\ 10^{-20} & -10^{-20} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$
$$= \begin{bmatrix} 10^{-20} & -10^{-20} & 1 \\ 10^{-20} & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 10^{-20} & 10^{-2} & 1 \\ 10^{-20} & 2 * 10^{-20} + 1 & 10^{-20} + 1 \\ 1 & 2 & 1 \end{bmatrix}$$

Novamente, vemos que o resultado não é exato, mas é muito próximo, tão próximo que acaba sendo igual após os arredondamentos.

Problema 6

Uma vez que você tem a decomposição LU de uma matriz quadrada A de ordem n (ou de PA, sendo P uma matriz permutação) a resolução de um sistema linear $Ax = b$ pode ser obtida mais rapidamente usando a decomposição LU já feita, em vez de fazer todo o escalonamento de novo.
Escreva uma função Scilab de nome Resolve˙com˙LU, que receba como variáveis de entrada uma matriz C com a decomposição LU de A (ou de PA, conforme matriz retornada pelas funções anteriores) e uma matriz B de ordem nxm e retorne uma matriz X, com a mesma ordem de B, cujas colunas sejam as soluções dos sistemas lineares $Ax_i = b_i$, $1 \leq i \leq m$.
Observação: talvez você ache necessário passar outra(s) variável(is) de entrada para essa função.
Teste a sua função com a matriz A_1 dada anteriormente e com a matriz B_1.
Teste também com a matriz A_2 dada anteriormente e com a matriz B_2.
Finalmente, teste com a matriz A_3 dada anteriormente e com a matriz $B_3 = B_2$.

$$B_1 = \begin{bmatrix} 2 & 4 & -1 & 5 \\ 0 & 1 & 0 & 3 \\ 2 & 2 & -1 & 1 \\ 0 & 1 & 1 & 5 \end{bmatrix} \quad B_2 = \begin{bmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Solução:

Explicação da função

Sejam, A, $P \in \mathbb{R}^{\ltimes\times\ltimes}$. X, Y, $B \in \mathbb{R}^{\ltimes\times>}$. Xi coluna de X de índice $i \in [\![1, m]\!]$.

Consideramos o sistema:

$$\begin{cases} AX = B \\ PA = LU \implies P^{-1}LUX = B \implies LY = PB \\ Ux = Y \end{cases}$$

Estabelecida essa relação, a função resolve, para cada coluna $LY_i = PBi$ com i variando de 1 a m.

Seja $Y_i(j)$ o j-ésimo elemento do vetor, ou seja, o elemento $Y(j, i)$ da matriz.

Resolvemos um sistema linear dado por uma matriz triangular inferiro, então é claro que se $j = 1$, obtemos $Y_i(1)$ por substituição direta, bastando calcular $\dfrac{B_i(1)}{L(1,1)}$. Como a diagonal de L é dada por $1's$ essa operação não tem efeito, mas no caso da retrosubstituição, quand lidamos com a triangular superior, realizar essa etapa é crucial para obter a solução correta da melhor forma possível.

$$\begin{bmatrix} L_{1,1} & L_{1,2} \\ L_{2,1} & L_{2,2} \end{bmatrix} \begin{bmatrix} Y_i(1) \\ Y_i(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{2,1} & 1 \end{bmatrix} \begin{bmatrix} Y_i(1) \\ Y_i(2) \end{bmatrix} = \begin{bmatrix} B_i(1) \\ B_i(2) \end{bmatrix} \implies Y_i(1) = \frac{B_i(1)}{1} \tag{25}$$

Para resolver o resto do sistema é só continuar a substituição. Assim chegamos em:

$$Y_i(2) = \frac{B_i(2) - (Yi(1) * L_{2,1})}{1} \tag{26}$$

Esse processo é instruido na função para resolver ambos os sitemas, $LY = B$ e então $UX = Y$. De forma geral temos que cada elemento é dado por:

$$Y_i(j) = \frac{B_i(j) - [(Y_i(1), \quad Y_i(2), \quad ... \quad , Y_i(j-1)) \cdot (L_{j,1}, \quad L_{j,2}, \quad ... \quad , L_{j,j-1})]}{L_{j,j}} \tag{27}$$

De forma análoga:

$$X_i(n) = \frac{Y_i(n)}{U_{n,n}} \tag{28}$$

$$X_i(j) = \frac{Y_i(j) - [(X_i(j), \quad X_i(j+1), \quad ... \quad , X_i(n)) \cdot (U_{j,j+1}, \quad U_{j,j+2}, \quad ... \quad , U_{j,n})]}{U_{j,j}} \tag{29}$$

Preparação para os testes

```
>> disp(B1)
     2     4    -1     5
     0     1     0     3
     2     2    -1     1
     0     1     1     5
```

```
>> disp(B2)
     1     1     2
     1    -1     0
     1     0     1
```

```
>> B3 = B2;
>> [x1, C1, P1] = Gaussian_Elimination_4(A1, b1);
>> [x2, C2, P2] = Gaussian_Elimination_4(A2, b2);
>> [x3, C3, P3] = Gaussian_Elimination_4(A3, b3);
```

Teste com A_1 e B_1

Nesse caso, as respostas são bem diferentes do que deveriam ser. A primeira solução (coluna 1) é exatamente o que deveria ser, contudo as demais são bem diferentes. No próximo vemos claramente que a única diferença se deve ao MATLAB desprezar o valor de 10^-20. Contudo, nesse caso não há nada do tipo. Talvez haja algum erro de aproximação devido às divisões entre float. Outra possibilidade é que há algum erro no código que justifique essa diferença, se esse for o caso, eu não fui capaz de achá-lo.

```
>> disp(Resolve_com_LU(C1, B1, P1))
     -2.0342       -2.067       1.5755        0.651
    -0.64957     -0.93875      0.60114     -0.29772
     0.54701       1.0712     -0.20798        1.584
     0.30769      0.10256     -0.17949     -0.35897
```

```
>> disp(inv(A1)*B1)
     -2.0342      -1.9316        1.453      0.81197
    -0.64957     -0.70085      0.60684      0.42735
     0.54701      0.90598     -0.24786       1.0085
     0.30769      0.38462    -0.076923      0.69231
```

Teste com A_2 e B_2

Novamente, as soluções (xi) obtidos diferem da solução real, provavelmente devido a alguma aproximação forçada do programa.

```
>> disp(Resolve_com_LU(C2, B2, P2))
```

```
     0      3      3
     0     -2     -2
     1      1      2

>> disp(inv(A2)*B2)
          0          3          3
     -1e-20         -2         -2
          1          1          2
```

Teste com A_3 e B_3

O resultado confere perfeitamente com o desejado.

```
>> disp(Resolve_com_LU(C3, B3, P3))
     0      3      3
     0     -2     -2
     1      1      2

>> disp(inv(A3)*B3)
     0      3      3
     0     -2     -2
     1      1      2
```

Funções

Gaussian_Elimination_2

```
% ////////////////////////////////////////////////////////////////////////////
% Variáveis de saída:
% x: solução do sistema Ax=b (assumimos que tal solução existe).
% C: Seja A=LU a decomposição LU de A.
% Então C(i,j)=L(i,j) para i>j e C(i,j)=U(i,j) para j>=i.
% ////////////////////////////////////////////////////////////////////////////
function [x, C, P] = Gaussian_Elimination_2(A, b)
C = [A, b];
[n] = size(C, 1);
P = eye(n);

for j = 1:(n-1)
  % O pivô deve estar na posição (j,j).

  % Se, em vez disso, tivermos um 0, trocamos a linha j por uma linha k,
  % sendo esta a primeira linha sem 0 na posição de pivô.
  if C(j, j) == 0
    % Acha a distância entre j e os elementos abaixo dele que têm valor 0.
    dist = find(C(j+1:n,j));

    % Caso onde a matriz A não é singular. Não existe decomposição LU.
    if dist(1) == 0
      error("O sistema é indeterminado");
    end

    % Atribui a k o índice da linha a ser trocada com j.
    k = dist(1) + j;

    % Há a troca das linhas k e j da matriz de permutação na decomposição PLU.
    P([j k], :) = P([k j], :);

    % Troca os elementos das linhas k e j da matriz completa [A|b]
    % a partir da coluna j, pois antes disso todos são 0.
    C([j k], j:n+1) = C([k j], j:n+1);
  end
```

```
  % Processo de decomposição LU de A.
  for i = (j+1):n
    % O elemento C(i,j) é o elemento na posição (i,j) de L na
    % decomposição LU de A.
    C(i, j) = C(i, j) / C(j, j);
    % Linha i <- Linha i - C(i,j)*Linha j.
    % Somente os elementos da diagonal ou acima dela são computados
    % (aqueles que compõem a matriz U).
    C(i, j+1:n+1) = C(i, j+1:n+1) - C(i, j) * C(j, j+1:n+1);
  end
end

x = zeros(n, 1);

% Calcula x, sendo Ux=C(1:n,n+1).

x(n) = C(n, n+1) / C(n, n);
for i = n-1:-1:1
  x(i) = (C(i, n+1) - C(i, i+1:n) * x(i+1:n)) / C(i, i);
end
C = C(1:n, 1:n);

end % Fim da função.
```

Gaussian_Elimination_3

```
% ///////////////////////////////////////////////////////////////////////
% Variáveis de saída:
% x: solução do sistema Ax=b (assumimos que tal solução existe).
% C: Seja A=LU a decomposição LU de A.
% Então C(i,j)=L(i,j) para i>j e C(i,j)=U(i,j) para j>=i.
% ///////////////////////////////////////////////////////////////////////
function [x, C, P] = Gaussian_Elimination_3(A, b)
C = [A, b];
[n] = size(C, 1);
P = eye(n);

for j = 1:(n-1)
  % O pivô deve estar na posição (j,j).

  % Se, em vez disso, tivermos um 0, trocamos a linha j por uma linha k,
  % sendo esta a primeira linha sem 0 na posição de pivô.
  if C(j, j) == 0
    % Atribui a dist o índice do elemento com maior valor absoluto dentre
    % os elementos da coluna j abaixo da linha j.
    [maior_pivo, dist] = max(abs(C(j+1:n,j)));

    % Caso onde a matriz A não é singular. Não existe decomposição LU.
    if maior_pivo == 0
      error("O sistema é indeterminado");
    end

    % Atribui a k o índice da linha com maior pivo em módulo.
    k = dist(1) + j;

    % Há a troca das linhas k e j da matriz de permutação na decomposição PLU.
    P([j k], :) = P([k j], :);

    % Troca os elementos das linhas k e j da matriz completa [A/b]
    % a partir da coluna j, pois antes disso todos são 0.
    C([j k], j:n+1) = C([k j], j:n+1);
  end
```

```

  % Processo de decomposição LU de A.
  for i = (j+1):n
    % O elemento C(i,j) é o elemento na posição (i,j) de L na
    % decomposição LU de A.
    C(i, j) = C(i, j) / C(j, j);
    % Linha i <- Linha i - C(i,j)*Linha j.
    % Somente os elementos da diagonal ou acima dela são computados
    % (aqueles que compõem a matriz U).
    C(i, j+1:n+1) = C(i, j+1:n+1) - C(i, j) * C(j, j+1:n+1);
  end
end

x = zeros(n, 1);

% Calcula x, sendo Ux=C(1:n,n+1).

x(n) = C(n, n+1) / C(n, n);
for i = n-1:-1:1
  x(i) = (C(i, n+1) - C(i, i+1:n) * x(i+1:n)) / C(i, i);
end
C = C(1:n, 1:n);

end % Fim da função.
```

Gaussian_Elimination_4

```
% ///////////////////////////////////////////////////////////////////////////
% Variáveis de saída:
% x: Solução do sistema Ax=b (assumimos que tal solução existe).
% C: Seja A=LU a decomposição LU de A.
% Então C(i,j)=L(i,j) para i>j e C(i,j)=U(i,j) para j>=i.
% P: Matriz de permutação da decomposição PLU de A.
% ///////////////////////////////////////////////////////////////////////////
function [x, C, P] = Gaussian_Elimination_4(A, b)
C = [A, b];
[n] = size(C, 1);
P = eye(n);

for j = 1:(n-1)
  % Queremos que na posição (j,j) haja o maior pivô possível. Para isso,
  % trocamos a linha j por uma linha k, sendo k o índice da linha cujo
  % elemento na posição (k,j) tem o maior valor em módulo.

  % Atribui a dist o índice do elemento com maior valor absoluto dentre
  % os elementos da coluna j abaixo da linha j.
  [maior_pivo, dist] = max(abs(C(j:n,j)));

  % Caso onde a matriz A não é singular. Não existe decomposição LU.
  if maior_pivo == 0
    error("O sistema é indeterminado");
  end

  % Atribui a k o índice da primeira linha abaixo de j com 0 abaixo de j.
  k = dist(1) + j - 1;

  % Há a troca das linhas k e j da matriz de permutação na decomposição PLU.
  P([j k], :) = P([k j], :);

  % Há a troca dos elementos das linhas k e j da matriz completa [A|b]
  % a partir da coluna j, pois antes disso todos são 0.
  C([j k], j:n+1) = C([k j], j:n+1);

```

```
  % Processo de decomposição LU de A.
  for i = (j+1):n
    % O elemento C(i,j) é o elemento na posição (i,j) de L na decomposição LU de A.
    C(i, j) = C(i, j) / C(j, j);
    % Linha i <- Linha i - C(i,j)*Linha j.
    % Somente os elementos da diagonal ou acima dela são computados
    % (aqueles que compõem a matriz U).
    C(i, j+1:n+1) = C(i, j+1:n+1) - C(i, j) * C(j, j+1:n+1);
  end
end

x = zeros(n, 1);

% Calcula x, sendo Ux=C(1:n,n+1).

x(n) = C(n, n+1) / C(n, n);
for i = n-1:-1:1
  x(i) = (C(i, n+1) - C(i, i+1:n) * x(i+1:n)) / C(i, i);
end
C = C(1:n, 1:n);

end % Fim da função
```

Resolve_com_LU

```
% ///////////////////////////////////////////////////////////////////////
%% Variáveis de entrada:
% B: Matriz (n x m).
% C: Seja A=LU a decomposição LU de A (n x n).
% Então C(i,j)=L(i,j) para i>j e C(i,j)=U(i,j) para j>=i.
%% Variáveis de saída:
% X: Matriz (n x m), cujas colunas sejam as soluções dos sistemas lineares
% Axi = bi, i <= i <= m.
% ///////////////////////////////////////////////////////////////////////
function [X] = Resolve_com_LU(C, B, P)
[n] = size(B, 1);
[m] = size(B, 2);
X = zeros(n, m);
Y = X;

% Criamos variáveis para armazenar explicitamente as matrizes L e U em C.
L = tril(C, -1) + eye(n);
U = triu(C);

% Seja A = (P.T)LU, Ux = y.
% Temos Ax = (P.T)Ly = b, Ly = Pb.
% Resolvemos primeiro Ly = Pb, então resolvemos Ux = y.

%% Resolvo Lyi = Pbi para achar yi
% As permutações necessárias são feitas entre as linhas da solução.
B = P * B;

% Acha o primeiro elemento por substituição direta.
Y(1, :) = B(1, :) / L(1, 1);

% Calcula todos os yi, sendo Axi = LUxi = Lyi = b
for g = 2:n
  Y(g, :) = (B(g, :) - dot(L(g, 1:g-1), Y(1:g-1, :))) / L(g, g);
end

%% Resolvo Uxi = yi, para achar xi
```

```
% Acha o primeiro elemento por retrosubstituição direta.
X(n, :) = Y(n, :) / U(n, n);

% Calcula todos os xi que são solução dos sistemas Uxi = yi.
for f = n-1:-1:1
  X(f, :) = (Y(f, :) - dot(U(f, f+1:n), X(f+1:n, :))) / U(f, f);
end
```